5-2017

Online/offline provable data possession

Yujue WANG
Singapore Management University, yjwang@smu.edu.sg

Qianhong WU

Bo QIN

Shaohua TANG

Willy SUSILO

DOI: https://doi.org/10.1109/TIFS.2017.2656461

Published in IEEE Transactions on Information Forensics and Security, Volume 12, Issue 5, May 2017, Article number 7828127, Pages 1182-1194.
http://doi.org/10.1109/TIFS.2017.2656461

Online/Offline Provable Data Possession

Yujue Wang, Qianhong Wu, *Member, IEEE*, Bo Qin, Shaohua Tang, *Member, IEEE*, and Willy Susilo, *Senior Member, IEEE*

***Abstract*— Provable data possession (PDP) allows a user to outsource data with a guarantee that the integrity can be efficiently verified. Existing publicly verifiable PDP schemes require the user to perform expensive computations, such as modular exponentiations for processing data before outsourcing to the storage server, which is not desirable for weak users with limited computation resources. In this paper, we introduce and formalize an *online/offline PDP* (OOPDP) model, which divides the data processing procedure into offline and online phases. In OOPDP, most of the expensive computations for processing data are performed in the offline phase, and the online phase requires only lightweight computations like modular multiplications. We present a general OOPDP transformation framework which is applicable to PDP-related schemes with *metadata aggregatability* and *public metadata expansibility*. Following the framework, we present two efficient OOPDP instantiations. Technically, we present aggregatable vector Chemeleon hash functions which map a vector of values to a group element and play a central role in the OOPDP transformation. Theoretical and experimental analyses confirm that our technique is practical to speed-up PDP schemes.**

***Index Terms*— Provable data possession, chameleon hash, online/offline signature, data outsourcing, cloud storage.**

I. Introduction

PROVABLE Data Possession (PDP) was introduced by Ateniese *et al.* [1] to secure data *integrity* in remote storage scenario, e.g., cloud storage. With PDP, the user is able to process her data to generate verifiable metadata, and deposit with a remote storage server, in such a way that the integrity can later be verified by challenging the server on a random set of metadata. That means with PDP, the user can enjoy data outsourcing services with data integrity guarantee, which can greatly save the user's local cost of storage space and maintenance. In this sense, PDP makes data outsourcing more attractive in real-world applications. Publicly verifiable PDP allows any people who holds the system public key to audit the outsourced data. It is interesting that these PDP schemes can provide efficient and flexible data sharing services with integrity guarantee in a multi-user setting, without leaking the private key of the file owner. However, in most publicly verifiable PDP-related schemes (e.g., [2]–[9]), the user has to perform a large number of modular exponentiations for processing a file. As in other primitives under the public key setting, exponentiations are relatively expensive computations for some users with limited computation resource. Thus, the expensive computations in PDP-related schemes will limit their wide usage especially for weak users, for example, on low-power devices such as wireless sensors (collecting data), smart phones and tablets.

To the best of our knowledge, there exists only one work [10] that devotes to speeding up PDP schemes. With their approach, the modular exponentiations are securely delegated to a powerful computation server. However, the user has to perform an exponentiation for processing one file block (in PDP-related schemes, the file is usually split into blocks and each block is processed separately). Also, the user has to interact with the computation server for outsourcing exponentiations in real time, which requires the computation server to be always available and incurs extra communication burden and delay. Hence, their approach does not allow the user to process a file by herself without resorting to an external entity. This motivates us to investigate a more practical way to speed up publicly verifiable PDP schemes.

A. Our Contributions

In this paper, we introduce the notion of *online/offline PDP* (OOPDP) and formalize its framework. In OOPDP, the file processing procedure is running in two phases, that is, an offline phase and an online phase. The first phase is carried out before knowing the file to be outsourced, where most expensive computations (e.g., modular exponentiations) in processing the file are performed. This phase can be executed when the user's computation device is idle or with the help from some computation server. Once a file is given, the

Manuscript received June 19, 2016; revised November 29, 2016; accepted December 20, 2016. Date of publication January 20, 2017; date of current version February 22, 2017. This work was supported in part by the Natural Science Foundation of China under Project 61672083, Project 61370190, Project 61532021, Project 61402029, Project 61472429, and Project 61632013, and in part by the Beijing Natural Science Foundation under Project 4132056. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Walid Saad.

Y. Wang is with the School of Information Systems, Singapore Management University, Singapore 188065, and also with the State Key Laboratory of Cryptology, Beijing 100878, China (e-mail: yjwang@smu.edu.sg).

Q. Wu is with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China, and also with the Network and Data Security Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu 610054, China (e-mail: qianhong.wu@buaa.edu.cn).

B. Qin is with the Key Laboratory of Data Engineering and Knowledge Engineering, Ministry of Education, School of Information, Renmin University of China, Beijing 100872, China, and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: bo.qin@ruc.edu.cn).

S. Tang is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: shtang@ieee.org).

W. Susilo is with the Centre for Computer and Information Security Research, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: wsusilo@uow.edu.au).

user in the online phase only needs to perform lightweight computations (e.g., modular multiplications) with the offline pre-computed results. The online phase can be performed very fast, which means OOPDP schemes are affordable by low-power devices.

We propose a semi-generic transformation framework from PDP to OOPDP. We identify two critical properties, metadata aggregatability and public metadata expansibility, for many existing publicly verifiable PDP schemes such as [2]–[8]. The former means that, every metadata contains an aggregatable component, which allows multiple metadata to be combined. In our transformation, the aggregatable metadata component in the underlying PDP is replaced by an aggregatable vector Chameleon hash function (AVCH), which permits the OOPDP user to produce intermediate aggregatable metadata for random values in the offline phase, and replace these random values by the actual file blocks in the online phase according to the Chameleon property of AVCH. The public metadata expansibility means that one can publicly and efficiently manipulate the metadata of a file block with a random element, without breaking the integrity of this block. Our semi-generic transformation is applicable to any public verifiable PDP-related schemes with the two properties, and the security of the transformed OOPDP can be reduced to the underlying scheme and AVCH.

In Section III, we formally define the notion of AVCH which extends the Chameleon hash function from [11]. In AVCH, a vector of values $\vec{m}$ and a random auxiliary parameter r are mapped to a single group element h. With the trapdoor key, the user can efficiently find a collision $(\vec{m}', r')$ such that $(\vec{m}', r')$ have the same hash value h as $(\vec{m}, r)$. AVCH satisfies the aggregatability property, which means that AVCH inputs and values can be respectively combined and the combined results still satisfy the mapping relation. We also present two concrete AVCH functions based on the discrete logarithm and factorizing assumptions. Note that AVCH has been used by Freeman [12] to design homomorphic signature schemes, without providing formal definition and security proof.

Under our transformation framework, we present two OOPDP instantiations based on the CDH and s-SDH assumptions, which are transformed from the Shacham-Waters' and Yuan-Yu's proofs of retrievability (PoR) schemes,[1] respectively (which are referred to as SW scheme and YY scheme in this paper). Theoretical analyses show that, in the OOPDP instantiations, the online procedure of processing a file only involves lightweight computations including modular multiplications and additions, and the overall computation and storage costs are comparable to the underlying PDP schemes. We also conduct thorough experiments, which indicate the file processing procedures in our OOPDP instantiations take only dozens of milliseconds when processing a 1MB file. The analyses further confirm that our OOPDP instantiations are useful for weak devices and can provide perfect user experience in practice.

[1]PoR is a slightly stronger notion than PDP in that PDP allows the user to check the integrity of outsourced data, while PoR additionally employs erasure code to tolerate data loss or corruption to some extent. In this paper, we only consider PDP schemes and PDP components in PoR schemes.

B. Related Work

1) Provable Data Possession: Outsourced data in clouds confront many security threats [13] such as integrity and privacy. To tackle the integrity issue, Ateniese *et al.* [1] introduced the notion of PDP. When auditing an outsourced file, neither the client needs to retrieve the whole file, nor the cloud storage server should touch the entire file. Wang *et al.* presented a data outsourcing scheme [8] that allows a third party auditor (TPA) to carry out integrity verification on behalf of the file owner, in the mean time TPA learns nothing of data contents. Wang *et al.* [6] presented a scheme in multi-user setting, where a security-mediator processes files for all users, but learns nothing about file contents. Recently, Yu et al. [14] presented a remote data integrity checking scheme in the identity-based setting, which offers zero knowledge privacy against a third party verifier. To support applications in a multiuser setting, an identity-based PDP scheme is presented from pre-homomorphic signatures in [15]. To cater for applications in multi-cloud storage scenario, Wang [16] designed a distributed PDP scheme to guarantee the integrity of outsourced files.

2) Proofs of Retrievability: Juels and Kaliski [4] introduced PoR to ensure the retrievability of outsourced files. Their construction supports finite rounds of integrity checking, since some special sentinels are inserted into the outsourced files for detecting the misbehavior of cloud server. Shacham and Waters [5] considered PoR in strong model and constructed privately and publicly verifiable PoR schemes. Yuan and Yu [9] investigated how to save communication costs in auditing the integrity, and proposed a public verifiable PoR scheme based on polynomial commitments [17]. Cui *et al.* [18] presented a publicly verifiable PoR scheme, which is proved secure against related-key attacks. Fan *et al.* [19] investigated data privacy against the auditor in remote storage schemes. The schemes proposed in [7] and [20] support updates on the outsourced data.

Note that privately verifiable PDP related schemes (e.g., [5, Sec. 3.2]) can be constructed from pseudo-random functions and MAC functions, which do not require expensive exponentiations for processing a file. However, privately verifiable schemes only allow some user who holds the private key to audit the outsourced file. That means these schemes cannot be directly deployed in the multi-user setting (e.g., for company application), where the outsourced data may need to be shared by a group of users. In fact, data sharing with the privately verifiable PDP related schemes would require the file owner to share her private key with all other file users, so as to audit the data integrity.

3) Online/Offline Cryptosystems: The notion of online/offline signature scheme (OOS) was introduced by Even *et al.* [21], where the signing algorithm is divided into an offline phase and an online phase. All the heavy computations of signing algorithm are carried out in the offline phase without knowing any message. In this way, the online phase can rapidly output a signature when the message to be signed is given. Even *et al.* [21] presented a general idea of transforming an ordinary signature scheme to an OOS version, i.e., by combining the ordinary signature

scheme with an efficient one-time signature scheme. Shamir and Tauman [22] developed a hash-sign-switch paradigm for realizing efficient OOS schemes with Chameleon hash [11]. Following this paradigm, several OOS schemes [23]–[28] and threshold OOS schemes [27], [29], [30] have been proposed. Gao *et al.* [31] studied divisible OOS in which the offline signature token can be exposed before seeing the message. Recently, the online/offline technique has been extended to the other crypto primitives, e.g., online/offline encryption [32]–[34], online/offline attribute-based encryption [35], and Γ-protocols [36]. However, to the best of our knowledge, there exists no result on online/offline PDP-related schemes in the literature.

C. Paper Organization

The remainder of this paper is organized as follows. Section II reviews the definition of PDP and covers some mathematical background. Section III defines aggregatable vector Chameleon hash function. Section IV introduces the framework and security model for OOPDP, and presents a semi-generic OOPDP transformation framework from existing PDP with some useful properties. Section V presents two OOPDP instantiations. The instantiations are analyzed and compared with the underlying PDP in Section VI. Finally, Section VII concludes the paper.

II. Preliminaries

In this section, we review the framework of PDP and the related complexity assumptions.

A. Provable Data Possession

A PDP scheme [1], [10] consists of five polynomial-time computable procedures, that is, KeyGen, $\mathsf{ProFile}$, Chall, PrfGen and Verify.

- $\mathsf{KeyGen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$: On input a security parameter $\kappa \in \mathbb{N}$, the (randomized) key generation procedure, which is carried out by cloud users, generates a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{ProFile}(\mathsf{pk}, \mathsf{sk}, M) \rightarrow (t, M^*)$: On input a public key pk, a secret key sk and a file $M \in \{0,1\}^*$, the file processing procedure, which is carried out by file owners, generates a file tag t and a processed file M^*. In this procedure, the given file M would be split into blocks $\vec{m}_i$ (which may be further split into sectors $m_{i,j}$), in such a way that a metadata σ_i will be produced for each block $\vec{m}_i$. All of metadata are contained in M^*.
- $\mathsf{Chall}(\mathsf{pk}, t) \rightarrow Q$: On input a public key pk and a file tag t, the challenge generation procedure, which is carried out by a verifier, produces a challenge Q.
- $\mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q) \rightarrow R$: On input a public key pk, a file tag t, a processed file M^* and a challenge Q, the proof generation procedure, which is carried out by the cloud storage server, produces a response R.
- $\mathsf{Verify}(\mathsf{pk}, t, Q, R) \rightarrow \{0,1\}$: On input a public key pk, a file tag t and a challenge-response pair (Q, R), the verification procedure outputs "1" if R is a valid response for Q, i.e., the challenged blocks are keeping intact; or "0" otherwise.

From now on, a round of *integrity verification protocol* refers to a sequential execution of procedures Chall, PrfGen and Verify on auditing some outsourced file, that is, the verifier runs $Q \leftarrow \mathsf{Chall}(\mathsf{pk}, t)$ and gives Q to the cloud server, who generates a proof $R \leftarrow \mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q)$ for the verifier to validate in $\mathsf{Verify}(\mathsf{pk}, t, Q, R)$. A PDP scheme should be *correct* in the sense that any outsourced file processed under any key pair must be successfully verified in any round of integrity verification protocol.

Correctness: A PDP scheme is *correct* if for any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and any file $M \in \{0,1\}^*$, let $(t, M^*) \leftarrow \mathsf{ProFile}(\mathsf{pk}, \mathsf{sk}, M)$, $\mathsf{Verify}(\mathsf{pk}, t, Q, \mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q)) = 1$ holds for any $Q \leftarrow \mathsf{Chall}(\mathsf{pk}, t)$.

The soundness of PDP scheme is modelled by the following security game, which is played by a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ with a challenger $\mathcal{C}$.

Setup: The challenger invokes $\mathsf{KeyGen}(1^\kappa)$ to generate a pair of public key and secret key $(\mathsf{pk}, \mathsf{sk})$ and gives pk to $\mathcal{A}$.

Queries: The adversary and the challenger jointly and adaptively carry out the following queries.

- *Processing file:* For each queried file M, the challenger $\mathcal{C}$ invokes $\mathsf{ProFile}$ to generate (t, M^*). The challenger sends (t, M^*) to $\mathcal{A}$ and maintains t.
- *Integrity verification:* In this type of queries, the challenger and the adversary play the role as a verifier and a prover, respectively. For any file M that has been queried for processing, the challenger can run $\mathsf{Chall}(\mathsf{pk}, t)$ to generate a challenge Q and send it to $\mathcal{A}$. The adversary returns a proof R. Then, the challenger verifies R by running $\mathsf{Verify}(\mathsf{pk}, t, Q, R)$ and gives the verification result to $\mathcal{A}$.

End-Game: Finally, the adversary outputs a description of a prover $\mathbb{P}$ corresponding to a file with tag t, i.e., the file has been queried for processing. The cheating prover $\mathbb{P}$ is *ε-admissible* if it can answer an ε fraction of challenges issued by some verifier.

Definition 1 (Soundness): [5] A PDP scheme is said to be *ε-sound* if there is an efficient extractor algorithm $Ext(\cdot)$ such that for any PPT adversary $\mathcal{A}$ who plays the above security game and outputs an ε-admissible cheating prover $\mathbb{P}$ for some file M, when given a pair of the public and secret keys $(\mathsf{pk}, \mathsf{sk})$, the file tag t and cheating prover $\mathbb{P}$, the extractor algorithm $Ext(\cdot)$ can recover M with overwhelming probability, i.e., $Ext(\mathsf{pk}, \mathsf{sk}, t, \mathbb{P}) = M$.

B. Mathematical Background

Let $G = \langle g \rangle$ be a cyclic group with prime order p. The group G is bilinear if there exists a cyclic group G_T and a bilinear map $\hat{e} : G \times G \rightarrow G_T$ that satisfy

- Bilinearity: $\forall \mu, \nu \in G$, and $\forall a, b \in Z_p^*$, $\hat{e}(\mu^a, \nu^b) = \hat{e}(\mu, \nu)^{ab}$;
- Non-degeneracy: $\hat{e}(g, g)$ is a generator of G_T;
- Efficiency: the map $\hat{e}$ and the group actions in G and G_T can be efficiently calculated.

The security of our schemes relies on the following computational assumptions.

1) Discrete Logarithm (DL) Assumption: Let $G = \langle g \rangle$ be a cyclic group with prime order p. Given a random element $h \in_R G$, any PPT algorithm $\mathcal{E}$ has only negligible probability in computing $x \in Z_p^*$ such that $h = g^x$.

2) Computational Diffie–Hellman (CDH) Assumption: Let $G = \langle g \rangle$ be a cyclic groups with prime order p. Given a triple $(g, g^\alpha, g^\beta) \in G^3$ where $\alpha, \beta \in_R Z_p^*$, any PPT algorithm $\mathcal{E}$ has only negligible probability in computing $g^{\alpha\beta}$.

3) s-Strong Diffie–Hellman (s-SDH) Assumption [37]: Let $G = \langle g \rangle$ be a cyclic groups with prime order p. Given a $(s+1)$-tuple $(g, g^z, \cdots, g^{z^s}) \in G^{s+1}$ where $z \in_R Z_p^*$, any PPT algorithm $\mathcal{E}$ has only negligible probability in computing a pair $(x, g^{\frac{1}{z+x}})$, where $x \in Z_p^* \setminus \{-z\}$.

III. Building Blocks

In this section, we extend the Chameleon hash function to aggregatable vector Chameleon hash (AVCH) function which plays a central role in our semi-generic OOPDP conversion. We also provide two efficient AVCH functions which will be used in our OOPDP instantiations.

A vector Chameleon hash scheme consists of three polynomial-time computable procedures, i.e., CKGen, CHash and TColl.

- $\mathsf{CKGen}(1^\kappa, s) \to (\mathsf{hk}, \mathsf{tk})$: On input a security parameter $\kappa \in \mathbb{N}$ and the size s of vectors, the (randomized) Chameleon hash key generation algorithm generates a pair of public hash key and secret trapdoor key $(\mathsf{hk}, \mathsf{tk})$.
- $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r) \to h$: On input a public hash key hk, a vector $\vec{m} \in_R \mathcal{M}^s$ and a random auxiliary parameter $r \in_R \mathcal{R}$, the Chameleon hash algorithm generates a hash value h.
- $\mathsf{TColl}(\mathsf{tk}, \vec{m}, r, \vec{m}') \to r'$: On input a secret trapdoor key tk, a pair of some vector and the corresponding auxiliary parameter $(\vec{m}, r) \in_R \mathcal{M}^s \times \mathcal{R}$, and another vector $\vec{m}' \in_R \mathcal{M}^s$, the trapdoor collision finding algorithm produces an auxiliary parameter $r' \in \mathcal{R}$ for $\vec{m}'$ such that $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r) = \mathsf{CHash}(\mathsf{hk}, \vec{m}', r')$.

A vector Chameleon hash function must satisfy the following properties:

- Collision resistance: There exists no PPT algorithm $\mathcal{E}$ which on input hk, outputs two distinct pairs $(\vec{m}, r)$ and $(\vec{m}', r') \in \mathcal{M}^s \times \mathcal{R}$ such that $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r) = \mathsf{CHash}(\mathsf{hk}, \vec{m}', r')$, with non-negligible probability.
- Indistinguishability: The distribution of r' is computationally indistinguishable from that of r.

A vector Chameleon hash function $\langle \mathsf{CKGen}, \mathsf{CHash}, \mathsf{TColl} \rangle$ is *aggregatable* if $h_1^{a_1} \cdot h_2^{a_2} = h$ holds, where $h_1 = \mathsf{CHash}(\mathsf{hk}, \vec{m}_1, r_1)$, $h_2 = \mathsf{CHash}(\mathsf{hk}, \vec{m}_2, r_2)$ and $h = \mathsf{CHash}(\mathsf{hk}, a_1\vec{m}_1 + a_2\vec{m}_2, a_1 r_1 + a_2 r_2)$, for all $s \in \mathbb{N}$, $(\mathsf{hk}, \mathsf{tk}) \leftarrow \mathsf{CKGen}(1^\kappa, s)$, $\vec{m}_1, \vec{m}_2 \in_R \mathcal{M}^s$, $a_1, a_2 \in_R Z_p^*$, and $r_1, r_2 \in_R \mathcal{R}$.

In a similar way, we can define an aggregatable hash function H_{Agg} without requiring the Chameleon property, that is, H_{Agg} does not has the TColl procedure and private trapdoor key tk.

A. DL-Based Aggregatable Vector Chameleon Hash

In [12], Freeman presented a type of DL-based AVCH without security proofs. We then review the function as follows. Suppose $G = \langle g \rangle$ is a cyclic group with prime order p.

- $\mathsf{CKGen}(1^\kappa, s)$: Choose a list of random values $x_1, \cdots, x_s \in_R Z_p^*$ and compute $u_i = g^{x_i}$ for every $1 \le i \le s$. Set $\mathsf{hk} = (p, g, u_1, \cdots, u_s)$ and $\mathsf{tk} = (x_1, \cdots, x_s)$.
- $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r)$: Denote $\vec{m} = (m_1, \cdots, m_s) \in (Z_p)^s$. Compute $h = g^r u_1^{m_1} \cdots u_s^{m_s}$.
- $\mathsf{TColl}(\mathsf{tk}, \vec{m}, r, \vec{m}')$: Denote $\vec{m} = (m_1, \cdots, m_s) \in (Z_p)^s$ and $\vec{m}' = (m_1', \cdots, m_s') \in (Z_p)^s$. Compute $r' = r + \sum_{i=1}^s x_i(m_i - m_i') \bmod p$.

Lemma 1: The above function $\mathsf{AVCH_{DL}}$ is an aggregatable vector Chameleon hash, under the DL assumption.

Proof: The efficiency, indistinguishability and aggregatability are straightforward, thus, we only prove the collision resistance property.

Suppose there is a PPT algorithm $\mathcal{E}$ which on input hk, outputs two pairs $(\vec{m}, r)$ and $(\vec{m}', r')$ such that $\vec{m} \neq \vec{m}'$ and $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r) = \mathsf{CHash}(\mathsf{hk}, \vec{m}', r')$, with non-negligible probability. Without loss of generality, assume that $m_1 \neq m_1'$ while $m_i = m_i'$ for all $2 \le i \le s$, which yields

$$r_1 + \sum_{j=1}^{s} x_j m_j = r_1' + \sum_{j=1}^{s} x_j m_j' \mod p.$$

It is easy to get x_1 by solving this equation, which contradicts the DL assumption. □

Remark 1: If choosing a random value $z \in_R Z_p^*$ and computing $u_i = g^{z^i}$ for each $i \in [1, s]$, then we can obtain a more efficient variant of $\mathsf{AVCH_{DL}}$. In this case, $\mathsf{tk} = z$ which means the secret size is greatly reduced. It is not difficult to validate that this variant does not degrade the security of the hash function.

Remark 2: Note that algorithm TColl is applied to a pair of vectors. In fact, it can also be applied to a vector and a message. Specifically, suppose $h = g^r u_1^{m_1} \cdots u_s^{m_s} \leftarrow \mathsf{CHash}(\mathsf{hk}, \vec{m}, r)$. Given a message $m' \in_R Z_p$, a collision can be computed as $r' = r + \sum_{i=1}^s x_i m_i - m' \bmod p \leftarrow \mathsf{TColl}(\mathsf{tk}, \vec{m}, r, m')$ such that $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r) = \mathsf{CHash}(\mathsf{hk}, m', r')$.

B. Factorizing Based Aggregatable Vector Chameleon Hash

- $\mathsf{CKGen}(1^\kappa, s)$: Randomly choose two large primes $p, q \in_R \{0,1\}^{\kappa/2}$ and compute $N = pq$. Pick a random value $g \in_R Z_N^*$ with order $\lambda(N)$ and a list of random values $x_1, \cdots, x_s$ that do not divide $\lambda(N)$, and compute $u_i = g^{x_i} \bmod N$ for every $1 \le i \le s$. Set $\mathsf{hk} = (N, g, u_1, \cdots, u_s)$ and $\mathsf{tk} = (x_1, \cdots, x_s)$.
- $\mathsf{CHash}(\mathsf{hk}, \vec{m}, r)$: Denote $\vec{m} = (m_1, \cdots, m_s) \in (Z_N)^s$. Compute $h = g^r u_1^{m_1} \cdots u_s^{m_s} \bmod N$.
- $\mathsf{TColl}(\mathsf{tk}, \vec{m}, r, \vec{m}')$: Denote $\vec{m} = (m_1, \cdots, m_s) \in (Z_N)^s$ and $\vec{m}' = (m_1', \cdots, m_s') \in (Z_N)^s$. Compute $r' = r + \sum_{i=1}^s x_i(m_i - m_i') \bmod \lambda(N)$.

Lemma 2: The above proposed $\mathsf{AVCH_F}$ is an aggregatable vector Chameleon hash, under the factorizing assumption.

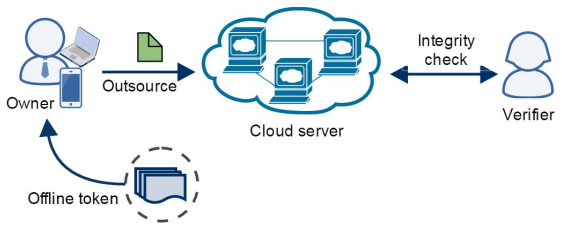


Fig. 1. The system model of OOPDP/PDP.

The proof is similar to Lemma 1 and omitted for avoiding repetition. Similarly, without degrading security, one can obtain a more efficient variant of $\mathsf{AVCH_F}$ by choosing a random value $z \in_R Z_N^*$ as tk and computing $u_i = g^{z^i} \mod N$ for each $i \in [1, s]$. Also, $\mathsf{AVCH_F.TColl}$ can be applied to a vector and a message as in $\mathsf{AVCH_{DL}}$.

IV. Semi-Generic Online/Offline PDP

A. Modelling OOPDP

As shown in Fig. 1, similar to PDP, an OOPDP system involves three entities: a file owner, a cloud storage server and a verifier. Both the file owner and verifier are cloud users. The file owner processes the file with offline tokens (which is not required in PDP) and sends the processed file to the cloud storage server. In this remote storage scenario, the server is not completely trusted by cloud users including file owner and verifier. The verifier is able to interact with the cloud server to audit the outsourced file with only public parameters.

Formally, an OOPDP scheme consists of six polynomial-time computable procedures, that is, KeyGen, $\mathsf{ProFile_{off}}$, $\mathsf{ProFile_{on}}$, Chall, PrfGen and Verify.

- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{PK}, \mathsf{SK})$: Similar to Section II-A.
- $\mathsf{ProFile_{off}}(\mathsf{PK}, \mathsf{SK}) \to \Sigma$: On input a pair of public key and secret key $(\mathsf{PK}, \mathsf{SK})$, the offline file processing procedure, which is carried out by the file owner, generates an offline file token Σ.
- $\mathsf{ProFile_{on}}(\mathsf{PK}, \mathsf{SK}, \Sigma, M) \to (t, M^*)$: On input a public key PK and a secret key SK, an offline file token Σ and a file $M \in \{0,1\}^*$, the online file processing procedure, which is carried out by the file owner, generates a file tag t and a processed file M^*. In this procedure, the given file M would be split into blocks $\vec{m}_i$ (which may be further split into sectors $m_{i,j}$), in such a way that a metadata σ_i will be produced for each block $\vec{m}_i$. All of metadata are contained in M^*.
- $\mathsf{Chall}(\mathsf{PK}, t) \to Q$: Similar to Section II-A.
- $\mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q) \to R$: Similar to Section II-A.
- $\mathsf{Verify}(\mathsf{PK}, t, Q, R) \to \{0,1\}$: Similar to Section II-A.

An OOPDP scheme should be *correct* in the sense that any outsourced file processed by consecutively offline and online phases under any key pair should be successfully verifiable in any round of integrity verification protocol.

Correctness: An OOPDP scheme is *correct* if for any $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and any file $M \in \{0,1\}^*$, let $\Sigma \leftarrow \mathsf{ProFile_{off}}(\mathsf{PK}, \mathsf{SK})$ and $(t, M^*) \leftarrow \mathsf{ProFile_{on}}(\mathsf{PK}, \mathsf{SK}, \Sigma, M)$, $\mathsf{Verify}(\mathsf{PK}, t, Q, \mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q)) = 1$ holds for any $Q \leftarrow \mathsf{Chall}(\mathsf{PK}, t)$.

Similar to Definition 1, we can define the soundness for an OOPDP scheme, except that the challenger consecutively invokes two procedures $\mathsf{ProFile_{off}}$ and $\mathsf{ProFile_{on}}$ to process a queried file in the security game. To avoid repetition, the security game and soundness definition are omitted here.

B. Useful Properties for OOPDP

Notice that most existing PDP-related schemes are designed in an *ad hoc* way, except a PoR scheme [38]. This situation makes it challenging to describe a generic OOPDP construction. Hence, in this paper, we focus on semi-generic transformation from some PDP to OOPDP. To achieve this goal, we first identify some useful properties of existing PDP-related schemes for building their OOPDP variations.

In existing PDP-related schemes, for achieving a tradeoff between the storage overhead and communication cost, the outsourced file M is usually split into a sequence of blocks $(\vec{m}_1, \cdots, \vec{m}_n)$, where each block comprises s ($s \geq 1$) sectors, i.e., $\vec{m}_i = (m_{i,1}, \cdots, m_{i,s})$, and the metadata is generated individually for each block. The file sectors $m_{i,j}$ are elements in Z_p or Z_N while the metadata σ_i are elements in a cyclic group G.

We observe that most existing PDP-related schemes (for example, [2]–[6], [8]) satisfy two properties: *metadata aggregatability* and *public metadata expansibility*.

- *Metadata Aggregatability*. The metadata for each file block has the following form
$$\sigma_i = (f(H(name\|i)) \otimes H_{Agg}(\vec{m}_i))^{\mathsf{k}} \tag{1}$$
where H is a collision-resistant hash function, *name* is the file name, $f : \{0,1\}^* \to G$ is some specific function, $\otimes$ is the group operation in G, $H_{Agg}(\vec{m}_i)$ denotes an aggregatable hash over a file block $\vec{m}_i$ and k is a secret key of the PDP scheme. This property is critical in presenting a semi-generic OOPDP transformation from PDP.
- *Public Metadata Expansibility*. A PDP scheme is publicly metadata expansible if there is a publicly computable efficient function Exp such that, for a metadata σ_i of any file block $\vec{m}_i$ and a random value $r_i \in_R Z_p^*$, it holds that
$$\begin{aligned}\mathsf{Exp}(\mathsf{pk}, \sigma_i, r_i) &= \sigma_i \otimes g_1^{r_i} \\ &= (f(H(name\|i)) \otimes H_{Agg}(\vec{m}_i) \otimes g^{r_i})^{\mathsf{k}}\end{aligned} \tag{2}$$
where g, g_1 are public parameters included in pk of the PDP scheme. Note that Exp is reversible, that is,
$$\mathsf{Exp}(\mathsf{pk}, \mathsf{Exp}(\mathsf{pk}, \sigma_i, r_i), -r_i) = \sigma_i \tag{3}$$
This property does not degrade the security of metadata and is critical to prove the security of the semi-generic OOPDP construction from PDP.

Remark 3: It is easy to check that most existing PDP-related schemes have an implicit function f, for example, $f(H(name\|i)) = H(name\|i)$ in Shacham and Waters' scheme [5], while $f(H(name\|i)) = u^{H(name\|i)}$ in Yuan and Yu's scheme [9].

C. Semi-Generic OOPDP Transformation from PDP

We propose a semi-generic OOPDP transformation which is applicable to any PDP with metadata aggregatability and public metadata expansibility. The key idea is that the aggregatable hash in the underlying PDP for composing metadata is replaced by an AVCH, and the standard signature scheme for producing file tags is replaced by an online/offline signature scheme. The Chameleon feature of AVCH allows the user to prepare offline matadata before knowing the file, and the online/offline signature scheme enables the user to produce an offline token for file tag in advance. Since all of expensive computations are performed in offline, the online computation complexity of the OOPDP scheme is as small as possible.

Let $\Pi = \langle \mathsf{KeyGen}, \mathsf{ProFile}, \mathsf{Chall}, \mathsf{PrfGen}, \mathsf{Verify} \rangle$ be a PDP scheme. Let $\Lambda = \langle \mathsf{CKGen}, \mathsf{CHash}, \mathsf{TColl} \rangle$ and $\Phi = \langle \mathsf{Keygen}, \mathsf{Sign}_{\mathsf{off}}, \mathsf{Sign}_{\mathsf{on}}, \mathsf{Verify} \rangle$ be an AVCH function and a secure OOS scheme, respectively, where the metadata in Π and hash values in Λ are in the same cyclic group G. Also, $\mathcal{M} = \mathcal{R} = Z^*$ in Λ. This way, we can get $H_{Agg}(\vec{m}_i) = \Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{m}_i, 0)$ for every file block $\vec{m}_i$ by setting appropriate parameters. Let B denote the bound of block number for any file to be outsourced, and let s be the sector number of a file block.

We describe an OOPDP scheme from Π as follows.

$\mathsf{KeyGen}(1^\kappa)$*:* Perform the following steps:

- Run $(\mathsf{opk}, \mathsf{osk}) \leftarrow \Phi.\mathsf{Keygen}(1^\kappa)$ of the OOS scheme.
- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{KeyGen}(1^\kappa)$ of the underlying PDP.
- Run $(\mathsf{hk}, \mathsf{tk}) \leftarrow \Lambda.\mathsf{CKGen}(1^\kappa, s)$ of the AVCH function.

Thus, $\mathsf{PK} = (\mathsf{opk}, \mathsf{pk}, \mathsf{hk})$ and $\mathsf{SK} = (\mathsf{osk}, \mathsf{sk}, \mathsf{tk})$.

$\mathsf{ProFile}_{\mathsf{off}}(\mathsf{PK}, \mathsf{SK})$*:* Compute the offline file tag token as follows

$$\Sigma_t \leftarrow \Phi.\mathsf{Sign}_{\mathsf{off}}(\mathsf{opk}, \mathsf{osk}) \tag{4}$$

Choose a random file name $name$. For each $1 \le i \le \mathsf{B}$, pick two random values $m_i', r_i' \in_R Z^*$, and compute an offline metadata token θ_i as in $\Pi.\mathsf{ProFile}$. The metadata token θ_i for m_i', r_i' has the following form:

$$\theta_i = \left(f(H(name\|i)) \otimes \Lambda.\mathsf{CHash}(\mathsf{hk}, m_i', r_i')\right)^{\mathsf{k}}$$

Then, store the offline file token $\Sigma = \{\Sigma_t, name\} \cup \{m_i', r_i', \theta_i\}_{1 \le i \le \mathsf{B}}$.

$\mathsf{ProFile}_{\mathsf{on}}(\mathsf{PK}, \mathsf{SK}, \Sigma, M)$*:* Split the file M into blocks such that each block has s sectors, that is,

$$M = \{\vec{m}_i = (m_{i,1}, \cdots, m_{i,s}) : 1 \le i \le n\} \tag{5}$$

where each sector $m_{i,j}$ is an element of some Z (e.g., Z_p or Z_N). Let $t_0 = name\|n$. Compute the file tag as follows

$$t \leftarrow t_0 \| \Phi.\mathsf{Sign}_{\mathsf{on}}(\mathsf{osk}, \Sigma_t, t_0) \tag{6}$$

For each file block $\vec{m}_i$ $(1 \le i \le n)$, compute

$$r_i = \Lambda.\mathsf{TColl}(\mathsf{tk}, m_i', r_i', \vec{m}_i)$$

This way, the metadata for block $\vec{m}_i$ is $\sigma_i = (\theta_i, r_i)$ and the final processed file is $M^* = \{\vec{m}_i, \sigma_i\}_{1 \le i \le n}$.

$\mathsf{Chall}(\mathsf{PK}, t)$*:* Run $\Phi.\mathsf{Verify}$ to validate t using opk. If it is invalid, output "0" and terminate; otherwise, same to the underlying PDP Π, pick a random subset $I \subseteq [1, n]$ and a random value $\upsilon_i \in_R Z^*$ for each $i \in I$. Send $Q = \{(i, \upsilon_i) : i \in I\}$ to the cloud storage server.

$\mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q)$*:* Run

$$R_\Pi = (\vec{\mu}, \theta) \leftarrow \Pi.\mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q)$$

where $\vec{\mu}$ is the combination for the challenged file blocks $\{\vec{m}_i\}_{i \in I}$ and θ is the aggregation for metadata tokens $\{\theta_i\}_{i \in I}$. Moreover, compute the combined auxiliary information r of $\{r_i\}_{i \in I}$ as $r = \Sigma_{i \in I} \upsilon_i r_i \in Z$. Then send $R = (\vec{\mu}, \sigma = (\theta, r))$ to the verifier.

$\mathsf{Verify}(\mathsf{PK}, t, Q, R)$*:* Similar to the underlying PDP Π. That is, parse R to obtain $\vec{\mu}$ and σ. If parsing fails, output "0" and terminate. Otherwise, check whether θ is valid for $\bigotimes_{i \in I} f(H(name\|i))^{\upsilon_i} \otimes \Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{\mu}, r)$ under pk as in $\Pi.\mathsf{Verify}$. Note that in $\Pi.\mathsf{Verify}$, θ is checked for $\bigotimes_{i \in I} f(H(name\|i))^{\upsilon_i} \otimes H_{Agg}(\vec{\mu})$ under pk. If so, output "1"; otherwise, output "0".

Remark 4: Similar to the public metadata expansibility of the underlying PDP scheme as shown in Equality (2), the cloud storage server is able to randomize every metadata. For example, for the i-th file blcok, the cloud storage server can randomly choose a value $a_i \in_R Z_p^*$ and compute

$$\theta_i' = \mathsf{Exp}(\mathsf{pk}, \theta_i, a_i) \text{ and } r_i' = r_i \cdot a_i \in Z$$

which implies

$$\theta_i' = (f(H(name\|i)) \otimes H_{Agg}(\vec{m}_i) \otimes g^{r_i'})^{\mathsf{k}}$$

Thus, the metadata in the transformed OOPDP scheme enjoy malleability. However, it is easy to see that malleability does not break or degrade the integrity of outsourced files.

Correctness: The integrity of the file tag is ensured by the OOS scheme Φ. Thus, we only show that the outsourced files are verifiable. According to the aggregatability of Λ, $\Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{\mu}, r)$ must be equal to the aggregation of $\{\Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{m}_i, r_i)\}$ of the challenged file blocks, where $\Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{m}_i, r_i) = \Lambda.\mathsf{CHash}(\mathsf{hk}, m_i', r_i')$ for every $i \in I$. Thus, θ and $\bigotimes_{i \in I} f(H(name\|i))^{\upsilon_i} \otimes \Lambda.\mathsf{CHash}(\mathsf{hk}, \vec{\mu}, r)$ satisfy the verification framework of $\Pi.\mathsf{Verify}$.

Theorem 1: Suppose the OOS scheme Φ for producing file tags is existentially unforgeable. If the underlying PDP scheme Π is sound and the AVCH function Λ is secure, then the above transformed OOPDP is sound for any PPT adversary $\mathcal{A}$.

Proof: We show that the soundness of the proposed OOPDP can be reduced to the underlying PDP Π and the AVCH function Λ. The basic idea is that the public metadata expansibility of Π allows the simulator to covert the integrity proof of the OOPDP attacker into a proof for PDP attacher, for answering the same integrity challenge. The following proof does not involve Φ since it is assumed secure and independent of the integrity of outsourced data.

Assume there is a PPT adversary $\mathcal{A}$ who can break the soundness of OOPDP, that is, it can output an ε-admissible cheating prover $\mathbb{P}$. The prover $\mathbb{P}$ should be able to forge a proof for some integrity challenge when auditing a file. Then, by

interacting with $\mathcal{A}$, we can construct a simulator $\mathcal{B}$ to output a forged proof for the same integrity challenge for the underlying PDP Π. In the following, $\mathcal{C}_\Pi$ denotes a challenger of Π.

Setup: For a security parameter κ, the simulator $\mathcal{B}$ requests $(\mathsf{pk}, \mathsf{sk})$ from $\mathcal{C}_\Pi$ and runs $\Lambda.\mathsf{CKGen}(1^\kappa, s)$ to get $(\mathsf{hk}, \mathsf{tk})$ for some positive integer s. Then $\mathcal{B}$ sends pk and hk to adversary $\mathcal{A}$.

Queries: The adversary $\mathcal{A}$ can adaptively interact with $\mathcal{B}$ in performing the following queries. The simulator $\mathcal{B}$ initiates an empty list to record all of intermediate information in processing file queries, i.e., all the queried files and responses.

- *Processing file:* For each queried file M from $\mathcal{A}$, the simulator $\mathcal{B}$ picks a random file name $name$, sends $(M, name)$ to $\mathcal{C}_\Pi$, and gets the corresponding metadata $\{\theta'_i : 1 \le i \le n\}$ as shown in Equation (1). Then, for each $1 \le i \le n$, the simulator $\mathcal{B}$ picks a random value $r_i \in_R Z$ and computes the metadata for OOPDP scheme as $\sigma_i = (\theta_i, r_i)$ where $\theta_i = \mathsf{Exp}(\mathsf{pk}, \theta'_i, r_i)$ as in Equality (2). The simulator gives the produced metadata $\{\sigma_i\}_{1\le i\le n}$ to $\mathcal{A}$.
- *Integrity verification:* In this type of queries, the simulator $\mathcal{B}$ and the adversary $\mathcal{A}$ play the roles as a verifier and a prover, respectively. For any file M that has been queried for processing, the simulator $\mathcal{B}$ can send a challenge $Q = \{(i, v_i) : i \in I\}$ to $\mathcal{A}$. The adversary returns a proof $\hat{R} = (\hat{\mu}_1, \cdots, \hat{\mu}_s, \hat{\sigma} = (\hat{\theta}, \hat{r}))$. The simulator verifies the proof $\hat{R}$ and gives the verification result to $\mathcal{A}$.

End-Game: At the end of the security game, the adversary $\mathcal{A}$ outputs a prover description $\mathbb{P}$ for a file M. The simulator $\mathcal{B}$ interacts with $\mathbb{P}$ to perform integrity verification protocol for file M. If $\mathcal{A}$ succeeds in attacking the soundness of OOPDP, then the prover $\mathbb{P}$ can forge a proof with probability at least ε at each round of integrity verification protocol. That means the proof $\hat{R} = (\hat{\mu}_1, \cdots, \hat{\mu}_s, \hat{\sigma} = (\hat{\theta}, \hat{r}))$ outputted by $\mathbb{P}$ is valid for the challenge Q, but differs from the proof $R = (\mu_1, \cdots, \mu_s, \sigma = (\theta, r))$ generated by the simulator from the maintained information for the same challenge. There are two cases to consider:

Case 1: $\hat{\theta} = \theta$.

In this case, $\hat{R} \neq R$ implies $(\hat{\mu}_1, \cdots, \hat{\mu}_s, \hat{r}) \neq (\mu_1, \cdots, \mu_s, r)$. On the other hand, $\hat{\theta} = \theta$ implies $\Lambda.\mathsf{CHash}(\mathsf{hk}, (\hat{\mu}_1, \cdots, \hat{\mu}_s), \hat{r}) = \Lambda.\mathsf{CHash}(\mathsf{hk}, (\mu_1, \cdots, \mu_s), r)$. Thus, the simulator breaks the AVCH function Λ.

Case 2: $\hat{\theta} \neq \theta$.

In this case, at least one pair of $\{(\hat{\mu}_j, \mu_j)\}_{1\le j\le s}$ or $(\hat{r}, r)$ should be different, otherwise $\hat{\theta} = \theta$ must hold. The simulator $\mathcal{B}$ computes $\hat{\theta}' = \mathsf{Exp}(\mathsf{pk}, \hat{\theta}, -\hat{r}) = \hat{\theta} \otimes g_1^{-\hat{r}}$ and $\theta' = \mathsf{Exp}(\mathsf{pk}, \theta, -r) = \theta \otimes g_1^{-r}$. We further distinguish two subcases.

Subcase 2.1: $\hat{\theta}' \neq \theta'$. In this case, both $(\hat{\mu}_1, \cdots, \hat{\mu}_s, \hat{\theta}')$ and $(\mu_1, \cdots, \mu_s, \theta')$ are valid proofs for the same challenge Q in Π. Thus, the simulator $\mathcal{B}$ obtains a forged proof $(\hat{\mu}_1, \cdots, \hat{\mu}_s, \hat{\theta}')$, which breaks the soundness of the underlying PDP scheme Π.

Subcase 2.2: $\hat{\theta}' = \theta'$. It implies that the adversary has re-randomized the outsourced metadata according to the malleability, but did not touch the outsourced file sectors. As stated in Remark 4, the integrity of outsourced file in OOPDP is not broken. Thus, the prover $\mathbb{P}$ fails to output a forged proof in attacking the soundness of OOPDP. □

D. Extension and Discussion

In our semi-generic OOPDP transformation framework, the file name is determined in the offline phase before the user knows the file. This is due to that our OOPDP transformation is designed as generic as possible and most expensive computations are performed in the offline phase. In fact, as we discussed in Section IV-B that in many existing PDP related schemes (e.g., [5]–[7]), $f(H(name\|i)) = H(name\|i)$. That means the metadata produced by these schemes have a component of exponentiation $(H(name\|i))^{\mathsf{k}}$, which is not replaceable in the online phase when it is contained in the offline metadata token.

The above discussed problem can be easily solved when transforming these PDP schemes into OOPDP ones. Specifically, during the transformation, the function is changed into $f(H(name\|i)) = u^{H(name\|i)}$ where u is a fixed value in cyclic group $G = \langle g \rangle$. (Notice that this type of function f has been used in [9].) In the corresponding OOPDP scheme, the user can randomly pick $x \in_R Z^*$ and compute $u = g^x \in G$. Here, Z^* would be Z_p^* or Z_N^* when this approach is used to construct OOPDP instantiations as shown in Section V. In fact, x and u can be seen as elements in tk and hk of an AVCH function, respectively. In this way, our OOPDP transformation can allow the user to choose a file name in the online phase.

The above discussed scenario is similar to ID-based online/offline encryption schemes [32], [34], [39], [40], where a specific identity is required in producing a ciphertext. But in the offline phase of encryption, the user may not know the identity. Lai *et al.* [39] observed that for some existing ID-based encryption schemes, a ciphertext has the form $CT = (\mathsf{Hdr}, C)$, where Hdr is an ID header and C is another component in ciphertext excluding Hdr. In $\mathsf{Hdr} = (g_1 g^{H(ID)})^s$, ID is a parameter in the exponent, where g, g_1 are public parameters, H is a hash function and $s \in Z_p^*$ is a random element chosen by encryptor. This formation of Hdr allows the user to generate offline components $C_1 = (g_1 g^\alpha)^s$ and $C_2 = g^{s\beta}$ for some random $\alpha, \beta \in_R Z_p^*$, and then in the online phase, the user only needs to compute $C_3 = \beta^{-1}(H(ID) - \alpha) \bmod p$ for some specific identity ID. Based on this observation, Lai *et al.* [39] presented a semi-generic transformation of ID-based online/offline encryption scheme, which is applicable to any ID-based encryption scheme such that the ciphertext has an ID header component.

V. Efficient OOPDP Instantiations

We now present two OOPDP instantiations in our semi-generic framework (see Section IV-C).

A. CDH-Based OOPDP Instantiation

The following instantiation is built from the CDH-based SW scheme [5]. Let $H : \{0,1\}^* \to G$ be a collision-resistant BLS hash function. Let Λ be a function of $\mathsf{AVCH}_{\mathsf{DL}}$.

$\mathsf{KeyGen}(1^\kappa)$*:* Carry out the following steps:

- Run $(\mathsf{opk}, \mathsf{osk}) \leftarrow \Phi.\mathsf{Keygen}(1^\kappa)$ of the OOS scheme.
- Pick a bilinear group $G = \langle g \rangle$ with prime order p and a bilinear map $\hat{e} : G \times G \rightarrow G_T$. Choose a random value $\mathsf{sk} = \alpha \in_R Z_p^*$ and compute $v = g^\alpha$.
- Run $\Lambda.\mathsf{CKGen}(1^\kappa, s)$ to get a public hash key $\mathsf{hk} = (p, g, u_1, \cdots, u_s)$ and a secret trapdoor key $\mathsf{tk} = (x_1, \cdots, x_s)$ of the AVCH function.

Thus, $\mathsf{PK} = (\mathsf{opk}, G, G_T, g, p, \hat{e}, v, u_1, \cdots, u_s)$ and $\mathsf{SK} = (\mathsf{osk}, \alpha, x_1, \cdots, x_s)$.

$\mathsf{ProFile}_{\mathsf{off}}(\mathsf{PK}, \mathsf{SK})$*:* Similar to Section IV-C, except that the file name *name* is randomly chosen from Z_p^*, and for every $1 \le i \le \mathsf{B}$, two random values $m_i', r_i' \in_R Z_p^*$ are picked and the offline metadata token θ_i is generated as follows:

$$\theta_i = (H(name\|i) \cdot g^{x_1 m_i' + r_i'})^\alpha = (H(name\|i) \cdot u_1^{m_i'} \cdot g^{r_i'})^\alpha$$

$\mathsf{ProFile}_{\mathsf{on}}(\mathsf{PK}, \mathsf{SK}, \Sigma, M)$*:* Similar to Section IV-C, except that the file M is split such that each sector $m_{i,j} \in Z_p$, and for every file block $\vec{m}_i$ $(1 \le i \le n)$, r_i is computed as follows:

$$r_i = m_i' x_1 + r_i' - \sum_{j=1}^{s} x_j m_{i,j} \mod p$$

$\mathsf{Chall}(\mathsf{PK}, t)$*:* Same to Section IV-C, except $v_i \in_R Z_p^*$ for every $i \in I$.

$\mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q)$*:* For each $1 \le j \le s$, compute

$$\mu_j = \sum_{i \in I} v_i m_{i,j} \in Z_p$$

Compute

$$r = \sum_{i \in I} v_i r_i \in Z_p, \quad \text{and} \quad \theta = \prod_{i \in I} \theta_i^{v_i} \in G$$

Send $R = (\mu_1, \cdots, \mu_s, \sigma = (\theta, r))$ to the verifier.

$\mathsf{Verify}(\mathsf{PK}, t, Q, R)$*:* If R cannot be parsed, output "0" and terminate. Otherwise, check whether

$$\hat{e}(\theta, g) \stackrel{?}{=} \hat{e}(\prod_{i \in I} H(name\|i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j} \cdot g^r, v)$$

If the condition holds, output "1"; otherwise, output "0".

Correctness: Since

$$\begin{aligned}
&\prod_{i \in I} H(name\|i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j} \cdot g^r \\
&= \prod_{i \in I} (H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}} \cdot g^{r_i})^{v_i} \\
&= \prod_{i \in I} (H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}} \cdot u_1^{m_i'} \cdot g^{r_i'} \cdot \prod_{j=1}^{s} u_j^{(-m_{i,j})})^{v_i} \\
&= \prod_{i \in I} (H(name\|i) \cdot u_1^{m_i'} \cdot g^{r_i'})^{v_i} \qquad (7)
\end{aligned}$$

we have

$$\begin{aligned}
\hat{e}(\theta, g) &= \hat{e}(\prod_{i \in I} (H(name\|i) \cdot u_1^{m_i'} \cdot g^{r_i'})^{v_i}, g^\alpha) \\
&= \hat{e}(\prod_{i \in I} H(name\|i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j} \cdot g^r, v)
\end{aligned}$$

According to Theorem 1, we have the following corollary.

Corollary 1: Suppose the OOS scheme Φ for producing file tags is existentially unforgeable. If the CDH-based SW scheme [5, Sec. 3.3] is sound, then the above transformed CDH-based OOPDP scheme is also sound for any PPT adversary $\mathcal{A}$.

Remark 5: The same idea can be applied to Wang *et al.*'s [7] scheme, where $s = 1$, which yields an online/offline dynamic PDP scheme.

In a similar way, we can get RSA-based OOPDP schemes by applying $\mathsf{AVCH}_{\mathsf{F}}$ to the RSA-based SW Scheme [5] and Ateniese *et al.*'s schemes [1], [41]. Note that in the security proof (i.e., reducing the soundness of RSA-based OOPDP to the underlying PoR), the simulator $\mathcal{B}$ cannot directly incorporate a random component as in the function Exp for producing an OOPDP metadata from a PDP metadata. This is due to that the Euler's totient function $\lambda(N)$ is a secret key of the RSA-based scheme. In the proof, we can circumvent this problem by choosing a random value and multiplying it by the public key e, and invoking Exp with this product.

In detail, in *processing file queries*, for each query file M from $\mathcal{A}$, $\mathcal{B}$ picks a random file name $name \in_R Z_N^*$, sends $(M, name)$ to $\mathcal{C}_\Pi$, and gets the corresponding metadata $\{\theta_i' : 1 \le i \le n\}$, where

$$\theta_i' = (H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^d \mod N.$$

Here, d is the private key in RSA-based OOPDP scheme. Then, for each $1 \le i \le n$, the simulator $\mathcal{B}$ randomly picks a value $\varsigma_i \in_R Z_N^*$, and computes the metadata for the OOPDP scheme as $\sigma_i = (\theta_i, r_i)$ where

$$\theta_i = \theta_i' \cdot g^{\varsigma_i} = (H(name\|i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^d \cdot g^{\varsigma_i} \mod N$$

and $r_i = e \cdot \varsigma_i$. The simulator gives the produced metadata $\{\sigma_i\}_{1 \le i \le n}$ to $\mathcal{A}$.

B. Optimized OOPDP Instantiation

In this section, we optimize the OOPDP instantiation proposed in Section V-A. Following the idea of Section IV-D, the resulting OOPDP instantiation allows the user to choose a file name in the online phase. Technically, we employ the polynomial commitment [17] and a variant of $\mathsf{AVCH}_{\mathsf{DL}}$ (see Remark 1) so that the communication costs for integrity auditing can be reduced to constant. In fact, the optimized OOPDP instantiation can also be seen as an online/offline version of YY scheme [9].

Let $H : \{0,1\}^* \rightarrow Z_p^*$ be a collision-resistant hash function. For a given vector $\vec{c} = (c_0, \cdots, c_s) \in Z_p^{s+1}$, let $f_{\vec{c}}(x) = \sum_{i=0}^{s} c_i x^i$ denote the polynomial over Z_p with coefficient vector $\vec{c}$.

$\mathsf{KeyGen}(1^\kappa)$*:* Carry out the following steps:

- Run $(\mathsf{opk}, \mathsf{osk}) \leftarrow \Phi.\mathsf{Keygen}(1^\kappa)$ of the OOS scheme.
- Pick a bilinear group $G = \langle g \rangle$ with prime order p and a bilinear map $\hat{e} : G \times G \rightarrow G_T$. Choose a random value $\mathsf{sk} = \alpha \in_R Z_p^*$ and compute $v = g^\alpha$.
- Run $\Lambda.\mathsf{CKGen}(1^\kappa, s+1)$ to get a public hash key $\mathsf{hk} = (p, g, u_1, \cdots, u_{s+1})$ and a secret trapdoor key $\mathsf{tk} = z$ of

TABLE I

PERFORMANCE COMPARISON OF CDH-BASED OOPDP WITH THE UNDERLYING SCHEME

	SW scheme [5, Section 3.3]	OOPDP scheme (Section 5.1)
ProFile computation	$n\mathtt{H} + ns\mathtt{M} + n(s+1)\mathtt{E} + n\mathtt{C}$	–
$\mathsf{ProFile}_{\mathsf{off}}$ computation	–	$\mathsf{B}\mathtt{H} + 2\mathsf{B}\mathtt{M} + 2\mathsf{B}\mathtt{E} + \mathsf{B}\mathtt{A} + 1\mathtt{C}_{off}$
$\mathsf{ProFile}_{\mathsf{off}}$ storage	–	$1\mathtt{E}_{oss} + (2\mathsf{B}+1)\|p\| + \mathsf{B}\mathtt{E}_G$
$\mathsf{ProFile}_{\mathsf{on}}$ computation	–	$n(s+1)\mathtt{A} + n(s+1)\mathtt{M} + 1\mathtt{C}_{on}$
Chall computation	$1\mathtt{C}_{vf}$	$1\mathtt{C}_{ovf}$
PrfGen computation	$((s+1)\|I\|-1)\mathtt{M} + \|I\|\mathtt{E} + s(\|I\|-1)\mathtt{A}$	$((s+2)\|I\|-1)\mathtt{M} + \|I\|\mathtt{E} + (s+1)(\|I\|-1)\mathtt{A}$
Verify computation	$2\mathtt{P} + \|I\|\mathtt{H} + (\|I\|+s-1)\mathtt{M} + (\|I\|+s)\mathtt{E}$	$2\mathtt{P} + \|I\|\mathtt{H} + (\|I\|+s)\mathtt{M} + (\|I\|+s+1)\mathtt{E}$
Communication costs (integrity auditing)	$(2\|I\|+s)\|p\| + 1\mathtt{E}_G$	$(2\|I\|+s+1)\|p\| + 1\mathtt{E}_G$

a variant of $\mathsf{AVCH}_{\mathsf{DL}}$ function discussed in Remark 1, that is, $u_i = g^{z^i}$ for each $i \in [1, s]$.

- Pick a random value $\beta \in_R Z_p^*$. Compute $\gamma = g^\beta$ and $\lambda = g^{\alpha z}$.

Thus, $\mathsf{PK} = (\mathsf{opk}, G, G_T, g, \hat{e}, v, u_0 = g, u_1, \cdots, u_{s+1}, \gamma, \lambda)$ and $\mathsf{SK} = (\mathsf{osk}, \alpha, z, \beta)$.

$\mathsf{ProFile}_{\mathsf{off}}(\mathsf{PK}, \mathsf{SK})$*:* Compute an offline file tag token Σ_t as in Equality (4). For each $1 \le i \le \mathsf{B}$, pick two random values $m_i', r_i' \in_R Z_p^*$, and compute an offline metadata token θ_i as follows:

$$\theta_i = (g^{xm_i' + r_i'})^\alpha = (u_1^{m_i'} \cdot g^{r_i'})^\alpha$$

Then, store the offline file token $\Sigma = \{\Sigma_t\} \cup \{m_i', r_i', \theta_i\}_{1 \le i \le \mathsf{B}}$.

$\mathsf{ProFile}_{\mathsf{on}}(\mathsf{PK}, \mathsf{SK}, \Sigma, M)$*:* Same to Section V-A, except that for every file block $\vec{m}_i$ $(1 \le i \le n)$, r_i is computed as follows:

$$r_i = m_i' z + r_i' - \beta H(name \| i) - \sum_{j=1}^{s} m_{i,j} z^{j+1} \mod p$$

$\mathsf{Chall}(\mathsf{PK}, t)$*:* Run $\Phi.\mathsf{Verify}$ to validate t with opk. If it is invalid, output "0" and terminate; otherwise, pick a random subset $I \subseteq [1, n]$, choose two random values $\tau, \rho \in_R Z_p^*$ and send $Q = \{\tau, \rho, I\}$ to the cloud storage server.

$\mathsf{PrfGen}(\mathsf{PK}, t, M^*, Q)$*:* For each $i \in I$, calculate $v_i = \rho^i \bmod p$. Compute $\vec{\mu} = (\mu_1, \cdots, \mu_s)$, r and θ in the same way as Section V-A. Define a polynomial

$$f_{\vec{\mu}}(x) = \sum_{j=1}^{s} \mu_j x^{j+1} \mod p$$

and calculate $y = f_{\vec{\mu}}(\tau)$. Then compute the polynomial

$$f_{\vec{\omega}}(x) = \frac{f_{\vec{\mu}}(x) - f_{\vec{\mu}}(\tau)}{x - \tau}$$

using polynomial long division. Denote the coefficient vector of $f_{\vec{\omega}}(x)$ as $\vec{\omega} = (\omega_0, \cdots, \omega_s)$. Compute

$$\psi = g^{f_{\vec{\omega}}(z)} = \prod_{j=0}^{s} (g^{z^j})^{\omega_j}$$

and send $R = (\psi, y, \sigma = (\theta, r))$ to the verifier.

$\mathsf{Verify}(\mathsf{PK}, t, Q, R)$*:* If R cannot be parsed, output "0" and terminate. Otherwise, check whether

$$\hat{e}(\theta, g) \stackrel{?}{=} \hat{e}(\gamma^{\sum_{i \in I} H(name \| i) \rho^i} \cdot g^{y+r} \cdot \psi^{-\tau}, v) \cdot \hat{e}(\psi, \lambda)$$

If the condition holds, output "1"; otherwise, output "0".

Correctness: Since

$$\begin{aligned}
\hat{e}(\theta, g) &= \hat{e}(\prod_{i \in I} (g^{zm_i' + r_i'})^{v_i \alpha}, g) \\
&= \hat{e}(\prod_{i \in I} (g^{\beta H(name \| i) + \sum_{j=1}^{s} m_{i,j} z^{j+1} + r_i})^{v_i \alpha}, g) \\
&= \hat{e}(\gamma^{\sum_{i \in I} v_i h_i}, v) \hat{e}(g^{\sum_{j=1}^{s} (\sum_{i \in I} v_i m_{i,j}) z^{j+1}}, v) \hat{e}(g^r, v) \\
&= \hat{e}(\gamma^{\sum_{i \in I} v_i h_i}, v) \hat{e}(g^{f_{\vec{\mu}}(z)}, v) \hat{e}(g^r, v)
\end{aligned}$$

and

$$\begin{aligned}
&\hat{e}(g^{y+r} \cdot \psi^{-\tau}, v) \cdot \hat{e}(\psi, \lambda) \\
&\quad = \hat{e}(g^{f_{\vec{\mu}}(\tau)}, g^\alpha) \hat{e}(g^r, v) \hat{e}(g^{-\tau f_{\vec{\omega}}(z)}, g^\alpha) \hat{e}(g^{f_{\vec{\omega}}(z)}, g^{\alpha z}) \\
&\quad = \hat{e}(g^r, v) \hat{e}(g^{f_{\vec{\mu}}(\tau) + (z - \tau) f_{\vec{\omega}}(z)}, g^\alpha) \\
&\quad = \hat{e}(g^r, v) \hat{e}(g^{f_{\vec{\mu}}(z)}, v)
\end{aligned}$$

the verification equation is satisfied.

For the security, we have the following corollary.

Corollary 2: Suppose the OOS scheme Φ for producing file tags is existentially unforgeable. If the YY scheme [9] is sound, then the above proposed OOPDP instantiation is also sound for any PPT adversary $\mathcal{A}$.

VI. PERFORMANCE

A. Theoretical Analysis

We summarize the comparisons between our OOPDP instantiations with the underlying schemes in terms of computation costs and offline storage costs in Tables I and II. In the tables, H, A, M, P and E denote a hash evaluation, an addition, a multiplication, a bilinear pairing and an exponentiation, respectively. Here, we do not differentiate the computations A, M and E on different groups or fields. $\mathtt{C}_{tag}$ represents the computation costs of the signature scheme for generating file tag in the underlying PDP schemes, while $\mathtt{C}_{on}$ and $\mathtt{C}_{off}$ respectively represent the online and offline computation costs of OOS for computing a file tag in OOPDP. We also respectively denote by $\mathtt{C}_{vf}$ and $\mathtt{C}_{ovf}$ the computation costs of running Verify of the regular signature scheme and OOS scheme for verifying a file tag. $\mathtt{E}_{oss}$ and $\mathtt{E}_G$ denote the sizes of an OOS signature and an element in bilinear group G, respectively. The computation cost of polynomial long division is denoted by $\mathtt{C}_D$. $|I|$ denotes the number of the challenged file blocks in Q.

Consider the OOPDP instantiation proposed in Section V-A and its underlying SW scheme [5, Sec. 3.3]. For generating a

TABLE II
PERFORMANCE COMPARISON OF s-SDH BASED OOPDP WITH THE UNDERLYING SCHEME

	YY scheme [9]	OOPDP scheme (Section 5.2)
ProFile computation	$n\mathtt{H} + n(s+1)\mathtt{M} + 3n\mathtt{E} + 1\mathtt{C}$	–
$\mathsf{ProFile}_{\mathsf{off}}$ computation	–	$2\mathsf{B}\mathtt{M} + \mathsf{B}\mathtt{E} + \mathsf{B}\mathtt{A} + 1\mathtt{C}_{off}$
$\mathsf{ProFile}_{\mathsf{off}}$ storage	–	$1\mathtt{E}_{oss} + 2\mathsf{B}\|p\| + \mathsf{B}\mathtt{E}_G$
$\mathsf{ProFile}_{\mathsf{on}}$ computation	–	$n\mathtt{H} + n(s+2)\mathtt{A} + n(s+2)\mathtt{M} + 1\mathtt{C}_{on}$
Chall computation	$1\mathtt{C}_{vf}$	$1\mathtt{C}_{ovf}$
PrfGen computation	$(s\|I\| + 2\|I\| + 3s - 3)\mathtt{M} + (\|I\| + s + 1)\mathtt{E}$ $+(s\|I\| - 1)\mathtt{A} + 1\mathtt{C}_D$	$(s\|I\| + 3\|I\| + 3s - 3)\mathtt{M} + (\|I\| + s + 1)\mathtt{E}$ $+((s+1)\|I\| - 2)\mathtt{A} + 1\mathtt{C}_D$
Verify computation	$\|I\|\mathtt{H} + 2(\|I\| + 1)\mathtt{M} + (\|I\| - 1)\mathtt{A} + 3\mathtt{E} + 3\mathtt{P}$	$\|I\|\mathtt{H} + 2(\|I\| + 1)\mathtt{M} + \|I\|\mathtt{A} + 3\mathtt{E} + 3\mathtt{P}$
Communication costs (integrity auditing)	$(\|I\| + 3)\|p\| + 2\mathtt{E}_G$	$(\|I\| + 4)\|p\| + 2\mathtt{E}_G$

metadata for a block with s sectors, PDP requires $(s+1)$ exponentiations, while the offline phase in OOPDP only needs to perform two exponentiations, which is independent of s. Thus, the offline computations of OOPDP are much more efficient than the file processing procedure in the SW scheme. In the online file processing phase of OOPDP, the user only needs to carry out efficient additions and multiplications. When auditing a file, an aggregated r of r_i associated with the challenged file blocks should be returned to the verifier, which means the procedure Verify in OOPDP would take one more exponentiation than that in the underlying SW scheme. Note that this OOPDP instantiation outperforms the offloaded file processing procedure for SW PDP scheme [10, Table 4], specifically, with the approach in [10], the user still has to perform n hash evaluations, $4n$ inversions and $(6ns + 1.5nlog\chi + 12n)$ multiplications, where χ is a security parameter.

As we noted before that the privately verifiable SW PDP scheme [5, Sec. 3.2] is also efficient for the user to process file. Specifically, to process a file with n blocks, their privately verifiable scheme needs to perform n evaluations on some pseudo-random function, ns multiplications and ns additions, where all of them are lightweight computations. Comparably, our OOPDP instantiation of the publicly verifiable SW scheme requires the user to carry out one online computation for the OSS scheme, $n(s+1)$ multiplications and $n(s+1)$ additions, in the online file processing phase. Both of them enjoy the same computation complexity in processing a file, which also implies our online/offline transformation technique greatly reduces the computation costs for the publicly verifiable SW scheme.

We continue to compare the s-SDH based OOPDP instantiation proposed in Section V-B with the underlying YY scheme [9]. We assume the secret values z^i $(2 \le i \le s+1)$ can be pre-computed by the user. For producing a metadata for a block with s sectors, the underlying PDP takes three exponentiations, while the offline file processing procedure in OOPDP only requires one exponentiation. The offline computations of OOPDP are more efficient than the file processing procedure in the YY scheme. In both cases, the number of exponentiations does not depend on the sector number s. When processing a file in the online phase in OOPDP, the user only needs to carry out efficient hash evaluation, additions and multiplications. In auditing an outsourced file, the aggregated challenged file blocks are not directly returned to the verifier. In fact, a polynomial evaluation and a commitment witness are sent to the verifier. Thus, in the s-SDH based PDP/OOPDP, the integrity auditing process only has constant communication complexity, while it requires the storage server to perform more computations than that in the CDH-based PDP/OOPDP. Here, the OOPDP instantiation also outperforms the offloaded file processing procedure for YY PDP scheme [10, Table 4], specifically, with the approach in [10], the user has to perform n hash evaluations, $4n$ inversions and $(ns + 1.5nlog\chi + 17n)$ multiplications, where χ is a security parameter.

In OOPDP, all tokens generated in the offline phase should be stored by the user, which size is linear to the (bound) number of file blocks. In either OOPDP instantiation, each metadata contains two elements θ_i and r_i. As shown in Sections IV and V, r_i is an auxiliary element introduced to speed up the online file processing procedure, which is not required in the underlying PDP scheme. Let l denote the sector size. For a file M of Ω bytes, the processed file by the OOPDP schemes contain $2\lceil\frac{\Omega}{s\times l}\rceil$ elements in G and Z_p, while the processed file by the underlying SW and YY schemes contain $\lceil\frac{\Omega}{s\times l}\rceil$ elements in G.

B. Experimental Analysis

We evaluate the performance of OOPDP instantiations and their underlying PDP schemes by conducting experiments with Pairing Based Cryptography library (PBC, http://crypto.stanford.edu/pbc/) in C programming language. The experiments are carried out on a system with Inter(R) Core(TM) i5-5200U CPU @ 2.20GHz and 2.20GHz processors, and 8.00GB RAM. The elliptic curve is of type $y^2 = x^3 + x$ with $|p| = 160$bits and $\mathtt{E}_G = 256$bits. Thus, the sector size is $l = 20$bytes.

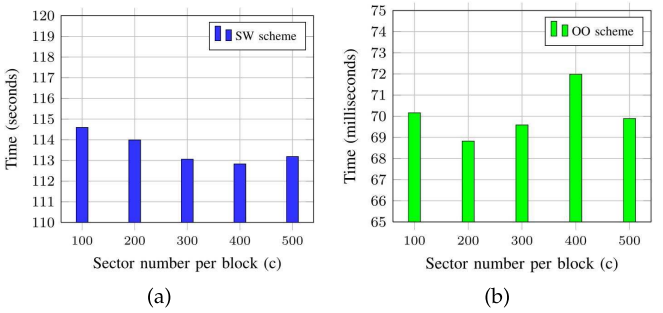
In the publicly verifiable SW scheme, processing a file M of Ω bytes requires in total $\mathsf{N}^{\mathsf{sw}}_{\mathsf{exp}}$ exponentiations, where

$$\mathsf{N}^{\mathsf{sw}}_{\mathsf{exp}} = n(s+1) \approx \left\lceil \frac{\Omega}{s \times l} \right\rceil \times (s+1) \tag{8}$$

Similarly, for processing the same file, the original YY scheme needs to perform $\mathsf{N}^{\mathsf{yy}}_{\mathsf{exp}}$ exponentiations

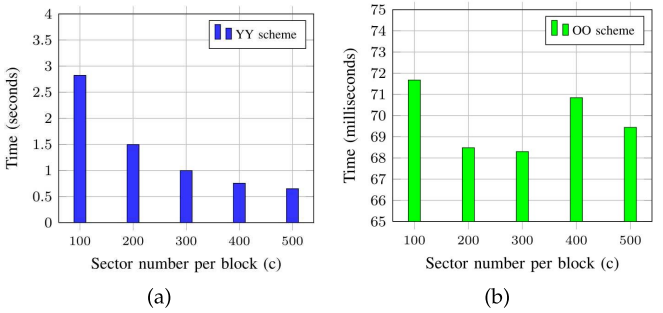
$$\mathsf{N}^{\mathsf{yy}}_{\mathsf{exp}} = 2n = 2\left\lceil \frac{\Omega}{s \times l} \right\rceil \tag{9}$$

Thus, the YY scheme is more efficient than the SW scheme when the file is split such that each block has more than one sector, i.e., $s > 1$. In our experiment, for comparing the efficiency, especially for showing the efficiency of OOPDP



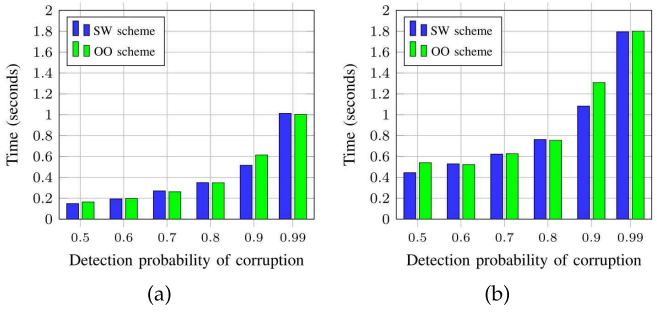
(a) (b)

Fig. 2. Processing a 1 MB file by (online/offline) SW scheme. (a) Underlying SW scheme. (b) Online/offline scheme.



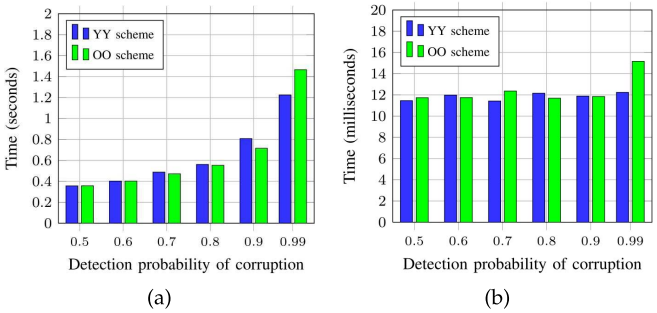
(a) (b)

Fig. 3. Processing a 1 MB file by (online/offline) YY scheme. (a) Underlying YY scheme. (b) Online/offline scheme.



(a) (b)

Fig. 4. Auditing a 1% corrupted file by (online/offline) SW scheme. (a) Generate proof. (b) Verification.



(a) (b)

Fig. 5. Auditing a 1% corrupted file by (online/offline) YY scheme. (a) Generate proof. (b) Verification.

schemes, we let all (original and online/offline) schemes to process the same file with $\Omega = 1$MB. For each scheme, we consider several cases of splitting the given file, that is, the number of sectors is set as $s = 100, \cdots, 500$. The number of required exponentiations can be calculated for each case according to Equations (8) and (9), which imply that the more sectors in a block, the more efficient for processing a file. Since the file tag generation depends on some specific digital signature scheme or OOS scheme, it is independent of file processing. Thus, the file tag generation and verification parts are omitted in the experiments. The simulation results of processing file are shown in Fig. 2 and Fig. 3, respectively, which demonstrate that our OOPDP instantiations only take about 70ms processing time. Moreover, we set $\mathsf{B} = 1000$ in the experiments of both OOPDP instantiations. For preparing B offline metadata tokens, online/offline SW and YY schemes take roughly 5.7s and 2.08s, respectively.

We continue to simulate the integrity auditing procedures. Notice that the OOPDP instantiations have the same probability P of detecting file corruption as the underlying PDP schemes, since the file is split in the same way in all schemes. As proved by Ateniese *et al.* [1], the probability P is determined by $|I|$. For an outsourced file with c percent randomly corruption, we have $P \approx 1 - (1-c)^{|Q|}$. In our experiment, suppose a file has been split such that each block has $s = 100$ sectors and has 1% corruption. The simulation results of (online/offline) SW and YY schemes at the cloud server and auditor are shown in Fig. 4 and Fig. 5. We see that the integrity proof requires constant verification time in both s-SDH-based OOPDP and the underlying YY scheme, for all cases with different detecting probability. Here, different detecting probability implies different number of file blocks are challenged, e.g., $P = 0.99$ requires $|I| \approx 460$ and $P = 0.9$ requires $|I| \approx 230$, etc. The experiments also indicate that the OOPDP instantiations do not incur any significant computation costs to the cloud server and auditor.

VII. Conclusion

In this paper, we presented an OOPDP framework which splits the file processing procedure into two phases. The offline phase performs almost all the heavy computations (e.g., modular exponentiations), which is run before knowing the file to be outsourced, and the online phase requires only lightweight computations (e.g., modular multiplications). We provided a generic transformation framework of converting PDP schemes with certain properties into OOPDP ones. In this framework, we presented two CDH-/s-SDH-based OOPDP instantiations from the existing PoR schemes. Theoretical analysis showed that in our OOPDP instantiations, not only the online file processing procedure can be performed very fast, but also the offline phase is much more efficient than the file processing procedure in the underlying schemes. The experimental analysis further confirms that our OOPDP instantiations provide perfect user experience for outsourcing data and are affordable by weak users, e.g., mobile devices with power supplies.

References

[1] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2007, pp. 598–609.

[2] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.*, Sep. 2008, Art. no. 9.

[3] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2009, pp. 213–222.

[4] A. Juels and B. S. Kaliski, Jr., "PoRs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2007, pp. 584–597.

[5] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, no. 3, pp. 442–483, 2013.

[6] B. Wang, S. S. M. Chow, M. Li, and H. Li, "Storing shared data on the cloud via security-mediator," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2013, pp. 124–133.

[7] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.

[8] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.

[9] J. Yuan and S. Yu, "PCPOR: Public and constant-cost proofs of retrievability in cloud," *J. Comput. Security*, vol. 23, no. 3, pp. 403–425, 2015.

[10] Y. Wang *et al.*, "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *Computer Security—ESORICS*, Germany: Springer-Verlag, vol. 8712, 2014, pp. 326–343.

[11] H. Krawczyk and T. Rabin, "Chameleon signatures," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)* 2000, pp. 143–154.

[12] D. M. Freeman, "Improved security for linearly homomorphic signatures: A generic framework," in *Public Key Cryptography—PKC*, vol. 7293, M. Fischlin, J. Buchmann, and M. Manulis, Eds. Germany: Springer-Verlag, 2012, pp. 697–714.

[13] K. Liang and W. Susilo, "Searchable attribute-based mechanism with efficient data sharing for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1981–1992, Sep. 2015.

[14] Y. Yu *et al.*, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 767–778, Apr. 2016.

[15] Y. Wang, Q. Wu, B. Qin, X. Chen, X. Huang, and J. Lou, "Ownership-hidden group-oriented proofs of storage from pre-homomorphic signatures," *Peer-to-Peer Netw. Appl.*, pp. 1–17, 2016. [Online]. Available: http://link.springer.com/article/10.1007%2Fs12083-016-0530-8

[16] H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Trans. Services Computing*, vol. 8, no. 2, pp. 328–340, Mar. 2015.

[17] A. Kate, G. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, vol. 6477, M. Abe, Ed. Germany: Springer-Verlag, 2010, pp. 177–194.

[18] H. Cui, Y. Mu, and M. H. Au, "Proof of retrievability with public verifiability resilient against related-key attacks," *IET Inf. Security*, vol. 9, no. 1, pp. 43–49, 2015.

[19] X. Fan, G. Yang, Y. Mu, and Y. Yu, "On indistinguishability in remote data integrity checking," *Comput. J.*, vol. 58, no. 4, pp. 823–830, 2015.

[20] J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 8, pp. 1717–1726, Aug. 2015.

[21] S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital signatures," in *Advances in Cryptology—CRYPTO*, vol. 435, G. Brassard, Ed. Germany: Springer-Verlag, 1990, pp. 263–275.

[22] A. Shamir and Y. Tauman, "Improved online/offline signature schemes," in *Advances in Cryptology—CRYPTO*, vol. 2139, J. Kilian, Ed. 2001, pp. 355–367.

[23] K. Kurosawa and K. Schmidt-Samoa, "New online/offline signature schemes without random oracles," in *Public Key Cryptography—PKC*, vol. 3958, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. 2006, pp. 330–346.

[24] X. Chen, F. Zhang, W. Susilo, and Y. Mu, "Efficient generic on-line/off-line signatures without key exposure," in *Applied Cryptography and Network Security*, vol. 4521, J. Katz and M. Yung, Eds. 2007, pp. 18–30.

[25] D. Catalano, M. Raimondo, D. Fiore, and R. Gennaro, "Off-line/on-line signatures: Theoretical aspects and experimental results," in *Public Key Cryptography—PKC*, vol. 4939, R. Cramer, Ed. 2008, pp. 101–120.

[26] P. Yu and S. R. Tate, "Online/offline signature schemes for devices with limited computing capabilities," in *Topics Cryptology—CT-RSA*, vol. 4964, T. Malkin, Ed. 2008, pp. 301–317.

[27] X. Chen *et al.*, "Efficient generic on-line/off-line (threshold) signatures without key exposure," *Inf. Sci.*, vol. 178, no. 21, pp. 4192–4203, 2008.

[28] X. Chen, F. Zhang, W. Susilo, H. Tian, J. Li, and K. Kim, "Identity-based chameleon hashing and signatures without key exposure," *Inf. Sci.*, vol. 265, pp. 198–210, May 2014.

[29] C. Crutchfield, D. Molnar, D. Turner, and D. Wagner, "Generic on-line/off-line threshold signatures," in *Public Key Cryptography–PKC*, vol. 3958, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. 2006, pp. 58–74.

[30] E. Bresson, D. Catalano, and R. Gennaro, "Improved on-line/off-line threshold signatures," in *Public Key Cryptography–PKC*, vol. 4450, T. Okamoto and X. Wang, Eds. 2007, pp. 217–232.

[31] C.-Z. Gao, B. Wei, D. Xie, and C. Tang, "Divisible on-line/off-line signatures," in *Topics Cryptology—CT-RSA*, vol. 5473, M. Fischlin, Ed. Germany: Springer-Verlag, 2009, pp. 148–163.

[32] F. Guo, Y. Mu, and Z. Chen, "Identity-based online/offline encryption," in *Financial Cryptography and Data Security—FC* (Lecture Notes in Computer Science), vol. 5143, G. Tsudik, Ed. Heidelberg, Germany: Springer-Verlag, 2008, pp. 247–261.

[33] J. K. Liu and J. Zhou, "An efficient identity-based online/offline encryption scheme," in *Applied Cryptography and Network Security*, vol. 5536, M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, Eds. Germany: Springer-Verlag, 2009, pp. 156–167.

[34] S. S. M. Chow, J. K. Liu, and J. Zhou, "Identity-based online/offline key encapsulation and encryption," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.*, New York, NY, USA, Mar. 2011, pp. 52–60.

[35] S. Hohenberger and B. Waters, "Online/offline attribute-based encryption," in *Public-Key Cryptography—PKC* (Lecture Notes in Computer Science), vol. 8383, H. Krawczyk, Ed. Heidelberg, Germany: Springer-Verlag, 2014, pp. 293–310.

[36] A. C. C. Yao and Y. Zhao, "Online/offline signatures for low-power devices," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 2, pp. 283–294, Feb. 2013.

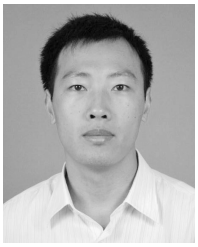
[37] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *J. Cryptol.*, vol. 21, no. 2, pp. 149–177, Apr. 2008.

[38] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Advances in Cryptology—"ASIACRYPT*, M. Matsui, Ed. Berlin, Germany: Springer 2009, pp. 319–333.

[39] J. Lai, Y. Mu, F. Guo, and W. Susilo, "Improved identity-based online/offline encryption," in *Proc. 20th Austral. Conf. Inf. Secur. Privacy (ACISP)*, 2015, pp. 160–173.

[40] J. Lai, Y. Mu, and F. Guo, "Efficient identity-based online/offline encryption and signcryption with short ciphertext," *Int. J. Inf. Secur.*, pp. 1–13, 2016. [Online]. Available: http://link.springer.com/article/10.1007/s10207-016-0320-6

[41] G. Ateniese *et al.*, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, May 2011, Art. no. 12.



Yujue Wang received the Ph.D. degrees from Wuhan University, Wuhan, China, and the City University of Hong Kong, Hong Kong, under the joint Ph.D. program, in 2015. He is currently a Research Fellow with the School of Information Systems, Singapore Management University. His research interests include applied cryptography, database security, and cloud computing security.

Qianhong Wu (M'15) received the Ph.D. degree in cryptography from Xidian University in 2004. Since then, he has been with Wollongong University, Australia, as an Associate Research Fellow, with Wuhan University, China, as an Associate Professor, and with Universitat Rovira i Virgili, Spain, as a Research Director. He is currently a Professor with Beihang University, China. He has been a holder/co-holder of nine China/Australia/Spain funded projects. He has authored over 28 patents and over 148 publications in leading journals and conferences. His research interests include cryptography, information security and privacy, VANET security, and cloud computing security. He is a member of IACR. He has served in the program committee of several international conferences in information security and privacy.

Bo Qin received the Ph.D. degree in cryptography from Xidian University, China, in 2008. Since then, she has been with the Xi'an University of Technology, China, as a Lecturer, and with Universitat Rovira i Virgili, Catalonia, as a Post-Doctoral Researcher. She is currently a Lecturer with Renmin University, China. Her research interests include pairing-based cryptography, data security and privacy, and VANET security. She has been a holder/co-holder of five China/Spain funded projects. She has authored over 80 publications in well-recognized journals and conferences and served in the program committee of a number of international conferences in information security.

Shaohua Tang (M'99) received the B.Sc. and M.Sc. degrees in applied mathematics, and the Ph.D. degree in communication and information system from the South China University of Technology, in 1991, 1994, and 1998, respectively. He has been a Full Professor with the School of Computer Science and Engineering, South China University of Technology, since 2004. His current research interests include information security, networking, and information processing.

Willy Susilo received the Ph.D. degree in computer science from the University of Wollongong, Australia. He is a Professor and the Head of the School of Computing and Information Technology with the University of Wollongong, Australia. He is also the Director of the Centre for Computer and Information Security Research with the University of Wollongong. He received the prestigious ARC Future Fellow by the Australian Research Council. His main research interests include cloud security, cryptography, and information security. He has served as a Program Committee Member in major international conferences and an Associate Editor of the number of prestigious international journals, including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY.